DAQ: Deadline-Aware Queue Scheme for Scheduling Service Flows in Data Centers

Cong Ding and Roberto Rojas-Cessa

***Abstract*—We propose a scheme to schedule the transmission of data center traffic to guarantee a transmission rate for long flows without affecting the rapid transmission required by short flows. We call the proposed scheme Deadline-Aware Queue (DAQ). The traffic of a data center can be broadly classified into long and short flows, where the terms long and short refer to the amount of data to be transmitted. In a data center, the long flows require modest transmission rates to keep maintenance, data updates, and functional operation. Short flows require either fast service or be serviced within a tight deadline. Satisfaction of both classes of bandwidth demands is needed. DAQ uses per-class queues at supporting switches, keeps minimum flow state information, and uses a simple but effective flow control. The credit-based flow control, employed between switch and data sources, ensures lossless transmissions. We study the performance of DAQ and compare it to those of other existing schemes. The results show that the proposed scheme improves the achievable throughput for long flows up to 37% and the application throughput for short flows up to 33% when compared to other schemes. DAQ guarantees a minimum throughput for long flows despite the presence of heavy loads of short flows.**

I. Introduction

Quickly responding to the users' requests is essential for the operation of a data center. The increasing number of data center applications relies on interactive soft real-time workloads that generate a large number of small requests. These requests generate partial responses that are collected and aggregated as a digest for the issuing users, in a timely fashion. These applications demand low latency for completion of short request/respond flows. Moreover, they have a unique and stringent deadline for receiving a response from the data center. For example, a typical data center application, Online Data Intensive (OLDI), has a soft-real-time constraint (e.g., 300 ms latency) and this specific requirement is recorded on service level agreements (SLAs), directing the application operation [1].

In general, the traffic generated by data center centric applications has the following characteristics: (1) Flows are delay sensitive, which means that the transmission of each flow is associated to a specific response time or deadline. If the provider fails to deliver the requested service by the deadline, the service is considered neglected and generates no revenues [2]. Therefore, complying with the deadlines associated with the received task requests is indispensable for data center sustainability. (2) Data centers use particular workflows that are enabled by a partition-aggregate model [3], [4]. This model for processing tasks, where a task is partitioned and processed by several processors for increasing performance or efficiency, is adopted by many web-based applications [5].

A number of recent works aim to reduce the number of flows missing their deadlines. One general approach is the reduction of flow transmission delay. DCTCP is a protocol that, as the Transmission Control Protocol (TCP), uses explicit congestion notification (ECN) and modifies senders' congestion window size in proportion to the perceived level of congestion. As a result, DCTCP reacts in time to traffic congestion and it guarantees a lossless transmission in the network. However, DCTCP is agnostic to deadline information and that makes 7% of the flows to miss their deadlines [4]. D^3 is a deadline-aware scheme that operates at the transport layer [4]. D^3 considers that once the flow is generated (by the application layer), the flow size (in number of bytes), and the deadline are all defined. Then the sender server requests a transmission rate (through a request message) to the switches in the path to the destination, including the minimum bandwidth required to make the flow finish transmission on time. Each switch manages these flow requests in a first-come first-serve (FCFS) fashion. Packets are queued in a first-in first-out (FIFO) queue with tail-drop, implemented on shared memory in these switches. In the case of a network with heterogeneous link capacities, the switch with the smallest output link capacity, called the bottleneck switch, determines the acceptable transmission rate assigned to flows passing through. However, FCFS may cause *deadline inversion*, which is the selection of requests from flows with not urgent deadlines instead of selecting those from flows with tight deadlines. Because D^3 considers no priorities for flows, it is then possible that urgent flows wait to receive service so as to miss their deadlines, causing degradation of the application throughput. Application throughput is defined as the number of flows that finish transmission within their deadline over the total number of flows requesting transmission bandwidth over a period of time. Degradation of application throughput jeopardizes the economic sustainability of the data center and, therefore, it must be avoided.

PDQ is a scheme that aims at both increasing application throughput and reducing the mean flow completion time [6]. It uses the earliest deadline first (EDF) and the shortest job first (SJF) policies. Supporting switches use these deadline-aware schemes in the implementation of PDQ.

Moreover, the traffic in data centers includes long background flows, in addition to the short flows [3], [4]. The long flows are maintenance data, carrying update information throughout the data center. Providing a minimum throughput to long flows is critical so as to guarantee that the data and

The authors are with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology. R. Rojas-Cessa is with the Networking Research Laboratory. Email: rojas@njit.edu

information within the data center is up to date, and at the same time, to preserve the value of using the data center. Therefore, a throughput for long flows, similar to what TCP would provide, must be accommodated. However, some schemes, such as PDQ, limit the service for long flows in exchange for service to short flows by given them higher priority (i.e., long flows may not be transmitted as long as short flows are to be transmitted). Moreover, the arrival rate of short flows is greater than that of long flows [3], which means that there is a chance that long flows may be starved from time to time. As a result, the information and storage value of the data center may be degraded.

There is, therefore, a need for a scheme that supports the transport layer in data center paths such that long flows achieve the needed transport data rate while making the maximum number of short flows finish transmission on time. These contending objectives need to be merged and satisfied.

To address that need, we propose the Deadline-Aware Queue (DAQ) scheduling scheme. DAQ is a transport-layer protocol that keeps high application throughput of short flows while guaranteeing sufficient throughput for long flows. DAQ also requires minimum infrastructure support and keeps no flow state at switches. DAQ satisfies the following performance requirements: (1) To meet deadlines for specific applications without recording the state of each flow, even at the times of bursty traffic in the network [4]. (2) To achieve high throughput for long background flows, independently of the load of the incoming short flows. Furthermore, the queue management is designed to assign priorities for deadline-constrained flows so as to avoid deadline inversion. This mechanism also improves the application throughput.

The remainder of this paper is organized as follows: Section II introduces the proposed scheme. Section III presents the performance of DAQ. Section IV presents our conclusions.

II. DAQ Scheduling Scheme

The DAQ scheme has the objective to maximize the number of short flows that finish transmission before their deadlines and yet to ensure that long flows receive the required bandwidth. A transmission control protocol for data center networks must:

- Maximize the number of flows completing transmission before deadlines
- Guarantee a high throughput for long flows.
- Allow high, if not 100%, link utilization.
- Achieve lossless transmissions.
- Minimize the amount of state information at switches

DAQ is aimed at satisfying these properties.

A. Overview

One would expect TCP to suffice for the transport of flows in a data center. However, the transmission of flows by TCP is lengthy and agnostic to flow properties [7]. There are four major reasons why TCP prolongs the flow completion time: (1) Use of the slow-start mechanism, which makes the transmission start with a small congestion window. The window size becomes large enough to provide high data rate after several round-trip times (RTTs). This phase holds data at the senders even when the network is capable of carrying them quickly. (2) It may allow long flows to monopolize the buffer and other switch resources because of the max-min fair-share rate that TCP exercises under cases of contention for bandwidth among different size flows [8]. (3) TCP may cause queue build up at the switch in case there are multiple flows converging and contending for the same switch output. (4) Use of timeouts and retransmissions for congestion avoidance and error control. TCP triggers recovering mechanisms after packet drops or retransmission timeouts (RTOs), and they take long time to recover achievable high transmission rates.

The DAQ scheme uses a mechanism that achieves high transmission rates in short time, called quick-start, at the senders and uses multiple service queues at switches. To overcome drawbacks (1) and (4) of TCP, DAQ adopts a link-by-link credit-based flow control [9]–[11]. This flow control mechanism works as follows: before sending any data to the network, every sender gets credits from the switch that is directly connected to the sender. A sender is allowed to transmit a total number of bytes equal to the number of credits. The largest number of credits the switch allocates to the sender equals the buffer size (or the portion of buffer allocated to the sender). In this way, flow control avoids packet loses and retransmissions.

Because the queuing delay at the switch is critical for providing fast service to short flows, and especially those whose deadline is small, the proposed scheme aims to make these flows leave the switch first by giving them the highest priority. DAQ uses two queues at supporting switches, one per flow type, to enable reaching a high transmission rate and avoiding memory monopolizing by larger flows. The queue for short flows is further divided into a queue for urgent short flows and another for not-urgent ones. Weighted round robin (WRR) scheduling is used to select which queue receives service and to guarantee a minimum level of service for each flow type. The weights used in WRR are split as a weight for short flows (w_s) and the other weight for long flows (w_l). The weight ratio, in this case, is defined as

$$WRR_r = \frac{short\ flow\ weight}{long\ flow\ weight} = \frac{w_s}{w_l}$$

In this way, DAQ addresses drawbacks (2) and (3) of TCP. Figure 1 shows the queue structure of the DAQ switch. Q_1 is the queue for short flows and Q_2 is the queue for long flows. The size of the Q_1 is small (and so is the bandwidth-delay product) as short flows require fast transmissions. Furthermore, Q_1 is divided into two priority queues: $Q_{1.1}$ for urgent short flows, used as the high priority queue, and $Q_{1.2}$ for not-urgent flows, used as the low priority queue. Therefore, $Q_{1.2}$ receives service only if $Q_{1.1}$ is empty. The use of these two priority queues avoids deadline inversion.

The urgency of a flow is determined by a deadline *threshold*. A flow is said to be urgent if $deadline \leq$ *threshold*, where $deadline$ is given as the amount of time left to finish the transmission of the flow, and the flow is forwarded to the urgent queue, $Q_{1.1}$. Otherwise, the short flow is forwarded to $Q_{1.2}$.

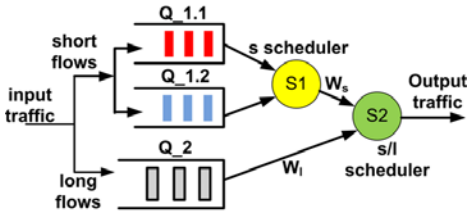


Fig. 1: Queue structure in DAQ-supporting switch: urgent flows use $Q_{1.1}$, not-urgent flows use $Q_{1.2}$, and long flows use Q_2. w_s is the weight for short flows and w_l is the weight for long flows.

TABLE I: Terminology

Name	Definition
RTT_{s-d}	Round-trip time between a worker (sender) and aggregator (destination)
RTT_{s-r}	Round-trip time between a worker (sender) and ToR switch
$sender\_win$	The largest number of bytes that each sender is allowed to send to the ToR switch. It is kept at the sender.
$flow\_win$	The number of bytes that can be sent per flow in one RTT_{s-d}, by each sender.
D	The time left before reaching the deadline. For example, D = 1 means that there is 1 ms left to transmit the remaining bytes of the flow.

The value of *threshold* must be smaller than the mean deadline of all flows coming into the destination server. If *threshold* is too small, very few packets are forwarded to the urgent queue and $Q_{1.2}$ builds up. This queue then may also have urgent flows, and these may be largely delayed. On the other hand, if *threshold* is too large, the urgent queue may queue some not-urgent flows and, as by product, affect some urgent flows. To avoid these two undesirable cases, we present a method to select a suitable *threshold* in Section III.

B. DAQ Algorithm

Table I introduces the terminology used in the remaining of this paper. DAQ follows two algorithms: Algorithm 1 describes the queue assignment at a ToR switch when a packet is received. Algorithm 2 describes a credit-based flow control mechanism used to avoid buffer overflow at the switch. Herein, we refer to the server that distributes tasks to other servers as *aggregator*, and those servers as *workers*. The transmission studied here is the one that occurs when workers send their responses (results) back to the aggregator. The ToR switch is placed between these two parties. The operation of DAQ is described by the following steps:

1) Downstream: aggregator $\rightarrow$ workers:

- The aggregator sends a query for performing an application request to the worker servers; the query's packet header carries the number of workers, N_w.

- The ToR switch receives the query and modifies the packet header by adding the value of $sender\_win$, where $sender\_win = \frac{buffer\_size}{N_w}$ for calculating the largest buffer size each worker may occupy. The switch forwards the query to every worker.

- Workers receive the query and initiate their $sender\_win$, accordingly.

2) Upstream: workers $\rightarrow$ aggregator:

- If a sender has a new flow ready for transmission, it puts a time stamp on the packet header, and sends it in a SYN packet to the destination (i.e., aggregator).

- The destination server responds with a SYN/ACK.

- Once the sender server gets the SYN/ACK message, it calculates the last RTT and updates the flow deadline with D = D-RTT. Then, data equal to $\min\{flow\_win,\ sender\_win\}$ are transmitted. $flow\_win$ is initiated at $\frac{bandwidth \times RTT_{s-r}}{2}$, and it adapts to the queue length being experienced at the switch.

- $sender\_win$ is updated by using processes in Lines 1 and 2 of Algorithm 2. The value of $sender\_win$ is reduced by one packet size (in bytes) every time a packet is sent.

- After receiving packets from multiple workers, the TOR switch applies Algorithm 1 to satisfy the different service requirements.

- When a packet leaves is forwarded to its destination, the operations stated in Lines 3-5 of Algorithm 2 are executed; the sender is informed of the available space in the buffer and the value of $sender\_win$ increases by one packet size (in bytes).

- The receiver sends an ACK for every data packet received.

- The sender uses the received ACK to calculate RTT_{s-d} and updates D. If a flow has remaining data for transmission ($left\_number\_bytes$), the sender transmits m bytes, where $m = \min\{flow\_win,\ sender\_win,\ left\_number\_bytes\}$

- Once the sender finishes transmitting the flow, a FIN packet is sent to the aggregator.

Because short flows are sensitive to queueing delays, the queue length must be kept within a suitable value. We apply a simple flow control, where $flow\_win = flow\_win + 1$, if the queue length (q) is smaller than the maximum queueing delay (Q), $q \leq Q$, and $flow\_win = \frac{flow\_win}{2^k}$, otherwise. Here, k is 1 if the incoming traffic load is smaller than or equal to 80% of the link capacity, and 2, otherwise.

Algorithm 1: Buffer management for packets

Input: A receiving packet P, round trip time RTT_{s-d}, Ratio is a positive integer
Output: The queue assignment for a receiving packet

```
// this packet belongs to a no-deadline flow
if P.deadline == -10 then
    push P into Q2
// this packet belongs to a deadline flow
else if P.deadline ≥ 0 then
    // set up the threshold
    T = Ratio*RTT_{s-d}
    if P.deadline ≤ T then
        push P into Q1.1
    else
        push P into Q1.2
```

Algorithm 2: Credit flow control

Input: A packet P, a sender S and a router R on the path
Output: Updating the $sender\_win$ at senders

```
// S sends P to the network
if P is a data packet then
    S.sender_win –;
// P leaves R's buffer
if P is a data packet then
    R sends notification to P.src(i.e. sender)
// S receives R's notification
// update S's sender window
S.sender_win ++;
```

III. Performance Evaluation

In this section, we evaluate the performance of the DAQ scheme and compare it to that of other existing schemes that hold similar implementation and operation complexity. In our simulations, we set the size of short flows randomly with a uniform distribution, in the interval [2 Kbyte, 58 Kbyte] and flow deadlines are randomly generated, with an exponential distribution, with a mean of 20 ms [4]. Figure 2 shows the simulation topology, which represents a typical rack in a data center network, where one server in the rack plays the role of an aggregator [12]–[14].

The amount of memory in the switch for packet buffering is set to the common capacity of 4 Mbyte, and the RTT between workers and destination server is 300 μs [4]. We compare DAQ with D^3 [4] and RCP [7], which are modeled and simulated on NS-2 [15].

We find the optimum value of *threshold* by looking into the application throughput. In the switch, we allocate 2 Mbytes to short flows and another 2 Mbytes to long flows. For a fair comparison between DAQ and the other two schemes, WRR is configured with WRR$r = 0.01$, which means 100 long-flow bytes are serviced per each short-flow byte.

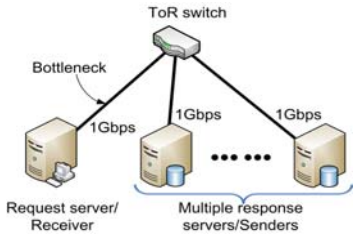


Fig. 2: Simulation topology.

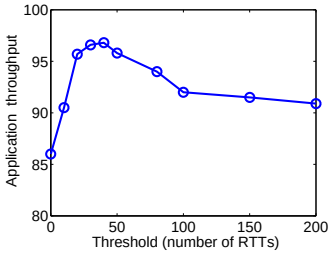


Fig. 3: The impact of *threshold* on application throughput.

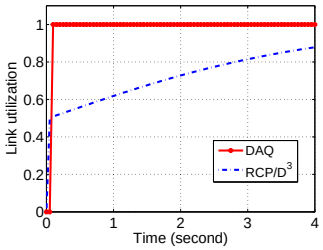


Fig. 4: Instantaneous normalized link utilization for 5 long flows.

A. DAQ Performance

1) Selection of threshold: To determine the appropriate *threshold* value, we simulated DAQ under different *threshold*s. The mean short-flow size is 30 Kbyte and each flow arrives with a Poisson distribution and an average rate of 3600 flows/s per switch.

Figure 3 shows the application throughput of short flows in function of $threshold$. The results show that when there is only one queue (*threshold*=0 or *threshold*=∞), the application throughput is about 86%, while the largest application throughput, 96%, is achieved for a *threshold* within [20, 60] RTTs. Furthermore, as *threshold* increases, the application throughput decreases as the urgent queue starts to get congested, and thus, blocking the most urgent flows. In summary, application throughput shows high sensitivity to *threshold*, and about 30 RTTs is an optimal value for *threshold*.

2) Link utilization: We generated five long flows from five distinct sources connected to the same ToR switch. The flows are destined to the same server. The simulation is run for 4.0 seconds. Because D^3 and RCP treat long flows similarly; they serve long flows as long as there is no short flow requesting bandwidth, we only compare RCP to DAQ. Figure 4 shows the link utilization of these two schemes produced by this experiment. As the figure shows, DAQ achieves nearly 1.0 link utilization during the complete simulation time while RCP achieves 0.5 utilization in the beginning and it gradually increases as time passes. The reason for this is that RCP estimates the explicit rate for the number of active flows

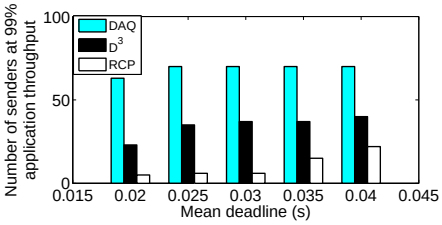


Fig. 5: Number of concurrent senders for achieving 99% application throughput with flow size mean of 10 Kbytes and mean deadline of 20 ms.

concurring at the switch. In contrast, the queues of DAQ help keeping the output link 100% utilized.

B. Data Center Workload

We evaluated DAQ under typical data center workloads. We measured the maximum application throughput achieved for short (and time sensitive) flows. For long flows, we measured: (1) the throughput during the presence of short flows, and (2) congestion under different traffic loads.

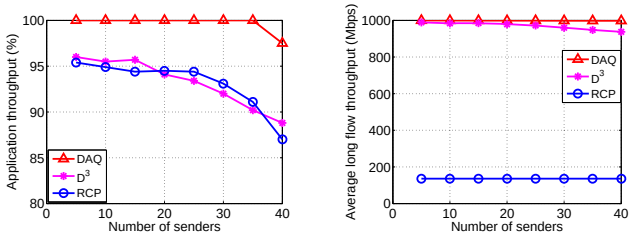
1) Incast Scenario: We first evaluate DAQ under an incast-like traffic scenario where several responses, comprising short flows with deadlines between 20 to 40 ms, converge to a ToR switch simultaneously. We measure the number of senders that can achieve 99% of application throughput. This experiment is similar to the one reported for D^3 [4]. We also measure the number of supported senders for RCP and D^3. Figure 5 shows that DAQ supports about 70 senders when 99% of application throughput is achieved. DAQ supports twice the number of senders D^3 supports, and more than three times that RCP does. Figures 6a to 6c show the application throughput achieved in the function of the number of senders, for three different average flow sizes, 10, 20, and 30 Kbyte. These three figures show that as the flow size increases, the application throughput of RCP and D^3 degrades rapidly. As an example, the application throughput of D^3 goes from 96 to 80% as the average flow size goes from 10 to 30 Kbytes, and the application throughput of RCP goes from 94 to 80% as the average flow size goes from 10 to 30 Kbytes. On the other hand, DAQ keeps the application throughput at 99% regardless of the flow size. However, under 30-Kbyte flows, the application throughput seems to be slightly lower than 99%. These results show that longer flows represent a more challenging situation as the spare time (i.e., the extra time from that used to transmit a flow) decreases. DAQ achieves higher application throughput than the other schemes because the urgent queue secures a fast service to urgent flows.

2) Background traffic: In this section, we test whether short and long flows can achieve their performance goals through the application of DAQ. For this, we create two scenarios: (1) Incast-like short response flows and two long flows, and (2) short flows with Poisson arrivals and two long flows. The size of a long flow is 100 Mbytes [3]. A sender generates a new long flow once the former flow finishes to keep the presence of long flows constant and for long time. The average short flow size is 15 Kbytes and the number of concurrent senders is varied from 5 to 40. The load of short flows is small, up to 0.3% the link capacity [16]. Figure 7a shows the application throughput of short flows. This figure shows that the application throughput of DAQ approaches to 100% for up to 35 senders, and it slightly decreases to about 98%, for 40 senders. In contrast, the application throughput of RCP and D^3 is about 95% for five senders and it drops to 90% for 40 senders. These results show that the presence of long flows does not affect DAQ, but it does affect RCP and D^3.

Figure 7b shows the achieved average throughput provided to long flows. The figure shows that among the three schemes, DAQ achieves the highest throughput. This is a result of isolating the small short-flow load and the large long-flow loads through WRR. The throughput of D^3 is slightly lower than that of DAQ and the difference is the result of the small short-flow load as D^3 provides service to long flows only in the absence of short flows.

We further evaluate the performance of the three considered schemes in scenario (2). Figure 8a shows the achieved application throughput of the three schemes. From this figure, we can see that all the schemes achieve high application throughput, above 98%, for arrival rates of up to 7000 flows/s. However, the application throughput of RCP and D^3 decreases to 40 and 30% for rates of 8000 flows/s, respectively. The application throughput of DAQ also decreases but it remains above 80%. Figure 8b shows the throughput of long flows. D^3 requires several RTTs to notify the sources and make the changed rate effective. RCP also requires time to converge to a fair-share bandwidth for estimating the accurate number of existing flows. RCP assigns initial bandwidth for all the flows during the 1.5 seconds that our simulation lasts, and the curve of RCP remains almost constant in the time period the figure shows.

IV. Conclusions

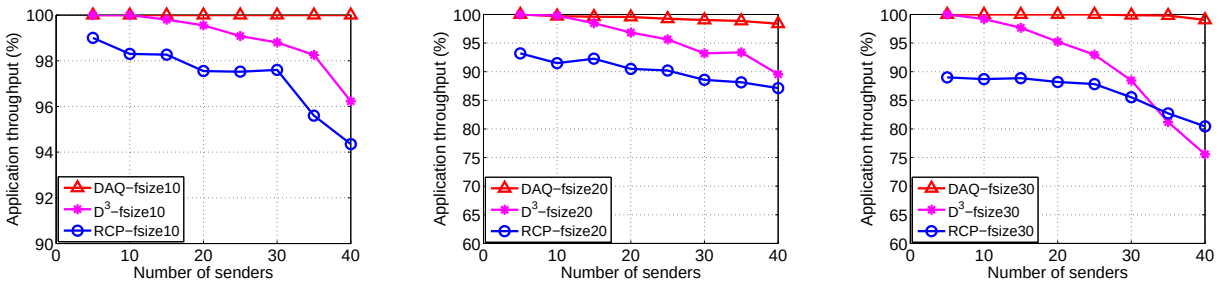


(a) Application throughput for short flows.

(b) Average long flows throughput.

Fig. 7: Performance of incast-like short flows and two background flows.

We introduced the deadline aware queue (DAQ) scheme to support the two large groups of data center traffic. One of the groups is denominated latency sensitive short flows. The other group is denominated long flows and they are instead sensitive to throughput. The proposed DAQ scheme uses different queues at supporting switches for the different

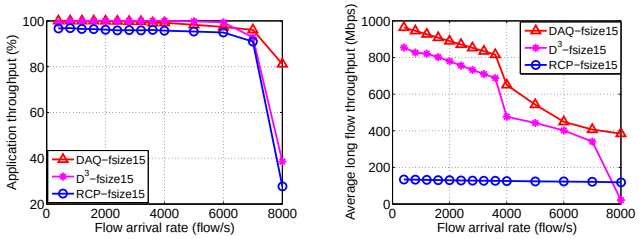


(a) Flow mean size of 10 Kbyte. (b) Flow mean size of 20 Kbyte. (c) Flow mean size of 30 Kbyte.

Fig. 6: Application throughput vs. number of senders. Mean deadline of short flows is 20 ms.

flow types and uses weighted round robin scheduling for determination of service to the different flows. Furthermore, we divide short flows into urgent and not-urgent to increase the number of short flows that finish service within their deadlines while keeping minimum state information. Urgent flows are served with higher priority than not-urgent flows. The deadline of a flow determines its urgency after it is compared to a reference *threshold* value. We show how to determine a suitable *threshold* value.

We performed a performance study through a simulation of data center traffic and the results show that the proposed scheme achieves high performance for short flows, measured in terms of application throughput under different traffic scenarios. Furthermore, DAQ provides high throughput to long flows, while providing high application throughput of short flows.



(a) Application throughput for short flows. (b) Average long flows throughput.

Fig. 8: Performance for Poisson arrivals short flows and two background flows.

REFERENCES

[1] A. Munir, I. A. Qazi, and S. B. Qaisar, "On achieving low latency in data centers," in *Proceedings of International Conference on Communications*. IEEE, 2013, p. 6.

[2] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 68–73, 2008.

[3] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," *Proc. Proc. ACM SIGCOMM*, vol. 40, no. 4, pp. 63–74, 2010.

[4] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," *SIGCOMM-*, vol. 41, no. 4, p. 50, 2011.

[5] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch, "Power management of online data-intensive services," in *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*. IEEE, 2011, pp. 319–330.

[6] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," *Proc. ACM SIGCOMM*, vol. 42, no. 4, pp. 127–138, 2012.

[7] N. M. Nandita Dukkipati, "Why flow-completion time is the right metric for congestion control and why this means we need new algorithms," *Proc. SIGCOMM*, vol. 41, no. 4, p. 50, 2006.

[8] S. B. Fred, T. Bonald, A. Proutiere, G. Régnié, and J. W. Roberts, "Statistical bandwidth sharing: a study of congestion at flow level," *Proc. ACM SIGCOMM*, vol. 31, no. 4, pp. 111–122, 2001.

[9] N. Kung and R. Morris, "Credit-based flow control for atm networks," *Network, IEEE*, vol. 9, no. 2, pp. 40–48, 1995.

[10] R. Rojas-Cessa, E. Oki, Z. Jing, and H. J. Chao, "Cixb-1: Combined input-one-cell-crosspoint buffered switch," in *High Performance Switching and Routing, 2001 IEEE Workshop on*. IEEE, 2001, pp. 324–329.

[11] R. Rojas-Cessa, E. Oki, and H. J. Chao, "Cixob-k: Combined input-crosspoint-output buffered packet switch," in *Global Telecommunications Conference, 2001. GLOBECOM'01. IEEE*, vol. 4. IEEE, 2001, pp. 2654–2660.

[12] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM*, vol. 38, no. 4. ACM, 2008, pp. 63–74.

[13] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *Proc. ACM SIGCOMM*, vol. 39, no. 4. ACM, 2009, pp. 51–62.

[14] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: a high performance, server-centric network architecture for modular data centers," *Proc. ACM SIGCOMM*, vol. 39, no. 4, pp. 63–74, 2009.

[15] http://www.isi.edu/nsnam/ns/, "ns-2 network simulator," 1989.

[16] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)," *Proc. ACM SIGCOMM*, vol. 42, no. 4, pp. 115–126, 2012.